Machine Learning Engineer Nanodegree

Capstone Project

Dina Nashaat
February 12th, 2019

I. Definition

Project Overview

The project explores Quora's insincere questions dataset using a Convolutional Neural Network approach to define if a given question is sincere or not.

This is a classical natural language processing problem, which takes language, in this case, text, as an input and outputs the corresponding classified result. In the following sections we will get into more technical detail on the approaches followed as well as the reported accuracy against a simple supervised learning approach as a base model.

CNN is usually used for image classification; however, it has proven to be effective for text classification as well.[i]

The project uses Quora's insincere questions dataset, which was available on Quora's Kaggle competition.

Problem Statement

The problem was held as a competition on Kaggle by Quora. Quora is an online platform where people post concise questions that can be answered by the community. A platform used by millions, and driven by the community is surely to suffer from insincere questions, those founded upon false premises, or that intend to make a statement rather than look for helpful answers, as described by Quora in their overview, in the competition page.

In this project, we will tackle this problem by developing a CNN model that can flag questions as insincere to detect toxic and misleading content with more scalable methods.

The problem is a classic NLP text classification problem, where a dataset of questions and posts are provided, and an insincere or not label, provided by manual review and machine learning.

Metrics

The metrics calculated are Accuracy, Precision, Recall and F1-Score. However, the main metric we will be comparing models with in this paper is going to be the F1-Score.

$$F_1 Score = harmonic\ Mean(precision, recall)$$

$$F_1 Score = 2\frac{Precision * Recall}{Precision + Recall}$$

As you will see in later sections, our dataset is unbalanced and there is a huge gap between the classes available in the dataset in terms of ratio, which means that the dataset is uneven. That is why the F1-Score presents a better approach than accuracy, because it takes false positives and false negatives into account, and it will always favor the less between Precision and Recall.

II. Analysis

Data Exploration

The dataset being used is Quora's insincere questions dataset. The dataset has only 1 feature (but later extracted to numerous), which is the question text and another column as target, namely 1 for insincere and 0 for sincere.

Two datasets were provided by the competition, the training dataset and testing dataset, however, in this project we will only use the training dataset which will be split into train, validation and testing, this is because the test dataset has no negative questions (label 1), which is going to be hard to evaluate if the classifier was doing a good job predicting abnormalities or not. Also, the training dataset was large enough to be split into train, validation and test.

Table 1 illustrates highlights which dataset was used and the size of each dataset provided, Table 2 highlights the data fields that were used for training and classification, and table 3 presents the characteristics of an insincere question as described by Quora on their competition page.

Table 1

Dataset		
Training Set	1.31 M x 3	A training set with 3 fields as shown in Table 2
Test set	56.4 K x 2	A test set with qid and question text
Sample Submission	56.4 K x 2	Submission file with qid and prediction

Table 2

Data Fields	
QID	Unique question identifier
Question Text	Quora question text
Target	Insincere has a value of 1 and 0 otherwise

Table 3

Insincere Phrase Characteristics

▪ Has a non-neutral tone
  • Has an exaggerated tone to underscore a point about a group of people
  • Is rhetorical and meant to imply a statement about a group of people
▪ Is disparaging or inflammatory
  • Suggests a discriminatory idea against a protected class of people, or seeks confirmation of a stereotype
  • Makes disparaging attacks/insults against a specific person or group of people
  • Based on an outlandish premise about a group of people
  • Disparages against a characteristic that is not fixable and not measurable
▪ Isn't grounded in reality
  • Based on false information, or contains absurd assumptions
▪ Uses sexual content (incest, bestiality, pedophilia) for shock value, and not to seek genuine answers

Data Samples:

	qid	question_text	target
389147	4c3bce0ece756675402c	What were the causes of the War of the Three Henrys?	0
1227398	f08a46796771b6c800af	Does a race car have the same turning radius as a regular car?	0
515252	64e2a9c3717931d7d803	Who did Lando win the Millenium Falcon from?	0
442120	56a402bd5da0a5f35a75	Do people that are autosomal recessive for a disease have parents that are both carriers for that disease?	0
208174	28b7e3369ea06b21ec71	With what purpose were the kites invented?	0

Statistics:

Total Training Set Records	`1,306,122`
Target 0 records count	`1,225,312`
Target 1 records count	`80,810`
Target 0 : Target 1	`15 : 1`
Test Set Split	`0.2 of Training set`

Exploratory Visualization

The dataset has no missing labels and is somehow well labeled, however this dataset has one main challenge, which is binary classes are imbalanced. Class 0: Class 1 is 15:1 as shown in figure 1. This presents a problem because the training could always favor the class with most records.

In the the following section, we mention in details how we tackled this problem using under-sampling and oversampling using SMOTE techniques.

Algorithms and Techniques

This project studies the effects of Convolutional Neural Networks on an NLP classification problem, and how it can improve the results over a classical supervised method

First, to handle problems with our imbalanced dataset, we have used under-sampling technique that reduces the number of records for the majority class and combined this method with SMOTE (Synthetic Minority over-sampling Technique), which is an oversampling technique that synthesizes the given data.

For preprocessing, some feature engineering was applied to the ‘question_text’ text feature, also I have used Word Embedding technique to vectorize sentences.

As for the classifier algorithms, Naïve Base, the supervised learning classifier, and a Convolutional Neural Network model have been designed to compare the results obtained.

Benchmark

The benchmark result is dependent on the Naïve Bayes classifier’s results. We take this model as the base to which we compare how well our CNN is doing and how much it has improved our model. The F1-Score recorded was various according to different preprocessing techniques, the results and justification sections, we illustrate more details on each training trial and the scores it has recorded as well as its performance.

III. Methodology

Data Preprocessing

Step 1: Imbalanced Dataset Refinement

The first thing to notice about the dataset is the huge imbalance between the two binary classes. The initial ratio in the original dataset between Class 0 (Sincere) and Class 1 (Insincere) is 15:1. This ratio is not a small one, and could pose a huge problem in over-fitting our data and biasing to one class only, the majority.

Since our dataset is large (1.3M), our first intuition was under-sampling; removing a percentage of Class 0, so that we can achieve a better ratio for Class 0 to 1 to 2:1 for example. This has indeed improved the accuracy for the Naïve Bayes model and has increased the score from 0.4 to 0.8, which is a huge leap.

Under-Sampling was an effective approach, however, we now have a huge amount of data which we are not putting into use, this is when another approach presented itself, which is use the SMOTE technique. Synthetic Minority over-sampling technique is an oversampling technique that synthesizes the minor data, and achieves approximately 1:1 in Class 0 to Class 1 ratio.

While experimenting, I used under-sampling technique alone, and then a combination between two methods was used because the dataset was huge, and synthesizing 15 times the number of records for class 0 might double the dataset size which might affect the training speed.

Step 2: Feature Engineering

Feature engineering refers to improving the text we have and converting it to a format to be fed to the neural network. To improve the text feature, we have to do some preprocessing on the text data we have, like formatting the text, removing special characters, lowering the text case and removing any stop-words that may occur. Also, we perform some stemming to the token to bring them back to their origin stem for example: ‘playing’ -> ‘play’ and so one, stemming bring a word back to a stem, but this stem might not be an English word, and in our context it is okay since this is data from the internet and probably contains a long of slang words not in the English dictionary.

Step 3: Vectorization

After converting our sentence to an array of tokens, we start by converting this to a vector space, numerical data that can be fed to the neural network.

Our records consist solely of textual data that cannot be used in any classifier as is, that is why we need a model like Bag of Words or word2Vec to map these words into vectors or numbers. In this project, we explore the use of Word Embedding.

While Bag of Words is a popular model that is associated most NLP tasks, and their use in classifiers like NB, it doesn't capture relation between words, it only records the number of occurrences or frequencies of each word, however, our dataset relies on the contextual meaning in this classification task, that is why it's better to use Word Embedding. That was especially used to be fed to the CNN, we used the GloVe (Global Vectors) word embedding for 100 dimensional vectors, which means it contains around 400,000 word, each word is represented by a 100 dimensional vector. GloVe has trained its vectors based on how frequently words appear together.

Implementation

Step 4: Naïve Bayes Model

After tokenizing sentences, and obtaining a vector space, we start by fitting the model on the Naïve Bayes model, we used this model primarily because it tends to fit well with text data, and as such, we're going to use this model as our base for comparison against the CNN model.

Step 5: Convolutional Neural Network

It took a lot of experimentation to tune the CNN hyper-parameters, and to prevent it from over fitting. Also, other than the hyper-parameters, the main challenge lies in our original dataset, and choosing the right ratio of target 0 class to target 1 class and running SMOTE to apply synthesized data on the dataset.

Refinement

In order to reach a satisfactory result, we have to tune the hyper-parameters of the CNN and the layers it comprises, also we need to refine the training set and the features selected. In the previous attempts, first we work with a training set that has balanced classes, by under-sampling the majority class by 90%. Then by tuning the parameters of the CNN, we reached a score close to that of the Naïve Bayes. However, the CNN model was over-fitting.

Next, we redefine the training set and under-sample the majority class only by 60%, and we mitigate the imbalance with the SMOTE technique. Using the same CNN network, the model has underperformed for Naïve Bayes, and was very slow to train for the CNN, so it was aborted.

Then in Attempt 2, we under-sample the majority class by 80%, and tune the hyper-parameter as the first attempt, however, the model underperforms and outputs a poor score of 0.16.

In the following attempts, we leave the training set balanced with under-sampling the majority class by 90% and we start by fine tuning the network through varying the class weights, experimenting with different optimizers, default `adam`, `custom adam` or experimenting with different loss function like `binary_crossentropy` or `categorical_crossentropy`.

We also experiment with `MaxPool`, `Dense` and `Conv1D` layers, in order to get close to the optimal model.

Finally, in Attempt 4 we see some slight improvement than the earlier models, but it turns out the number of epochs was very small, so in attempt 5 we increase the number of epochs to 100 which gives us the closest accuracy there is to Naïve Bayes, with even more hyper-parameters tuning, attempt 6 does not yield a better result.

IV. Results

Model Evaluation and Validation

Attempt 1:

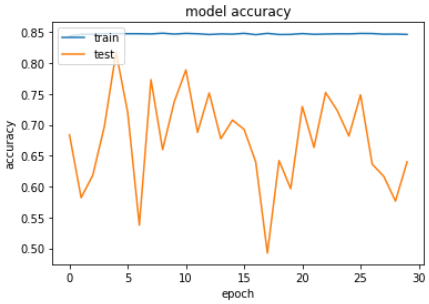
a) **Preprocessing:**
   a. Under-Sampling the majority class to only 0.1 of the its size
   b. Feature Engineering applied
   c. Class weights adjusted to favor Class (1) 3 times Class 0

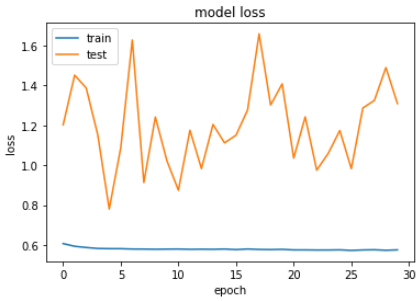
b) **Model Implementation:**
   - Number of Epochs: 30

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, 161, 100)          4416000
_________________________________________________________________
dense_28 (Dense)             (None, 161, 64)           6464
_________________________________________________________________
dropout_21 (Dropout)         (None, 161, 64)           0
_________________________________________________________________
dense_29 (Dense)             (None, 161, 128)          8320
_________________________________________________________________
dropout_22 (Dropout)         (None, 161, 128)          0
_________________________________________________________________
flatten_10 (Flatten)         (None, 20608)             0
_________________________________________________________________
dense_30 (Dense)             (None, 1)                 20609
=================================================================
Total params: 4,451,393
Trainable params: 35,393
Non-trainable params: 4,416,000
_________________________________________________________________
None
```

c) **Results:**

CNN Training Accuracy and Loss





Attempt 2:

a) Preprocessing:

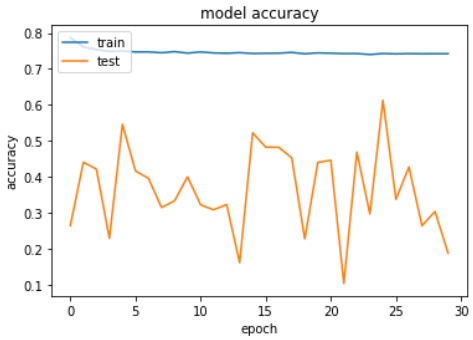
a. Under-Sampling the majority class to only 0.2 of the its size
b. Feature Engineering applied
c. Class weights adjusted to favor Class 1 15 times Class 0

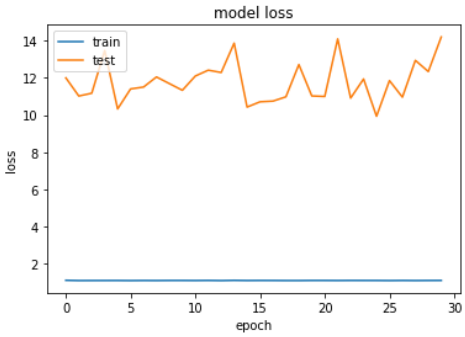
b) Model Implementation

- Number of Epochs: 30

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 161, 100)          5828600
_________________________________________________________________
dense_26 (Dense)             (None, 161, 64)           6464
_________________________________________________________________
dropout_19 (Dropout)         (None, 161, 64)           0
_________________________________________________________________
flatten_9 (Flatten)          (None, 10304)             0
_________________________________________________________________
dropout_20 (Dropout)         (None, 10304)             0
_________________________________________________________________
dense_27 (Dense)             (None, 1)                 10305
=================================================================
Total params: 5,845,369
Trainable params: 16,769
Non-trainable params: 5,828,600
_________________________________________________________________
None
```

c) Result





Attempt 3:

a) Preprocessing:

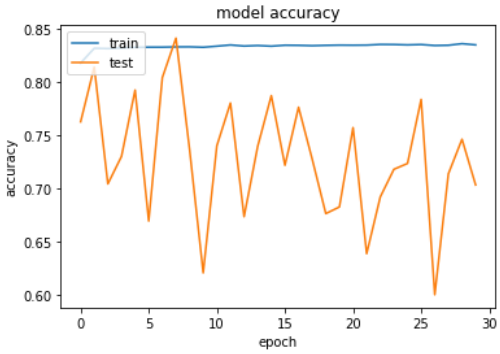
a. Under-Sampling the majority class to only 0.1 of the its size
b. Feature Engineering applied
c. Class weights adjusted to favor Class 1 3 times Class 0

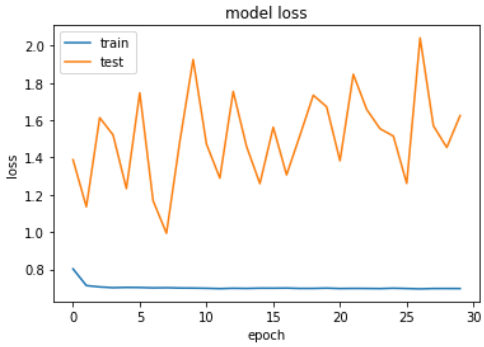
b) Model Implementation

- Number of Epochs: 30

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, 161, 100)          4416000
_________________________________________________________________
dense_45 (Dense)             (None, 161, 16)           1616
_________________________________________________________________
dropout_38 (Dropout)         (None, 161, 16)           0
_________________________________________________________________
dense_46 (Dense)             (None, 161, 32)           544
_________________________________________________________________
dropout_39 (Dropout)         (None, 161, 32)           0
_________________________________________________________________
flatten_16 (Flatten)         (None, 5152)              0
_________________________________________________________________
dense_47 (Dense)             (None, 1)                 5153
=================================================================
Total params: 4,423,313
Trainable params: 7,313
Non-trainable params: 4,416,000
_________________________________________________________________
None
```

c) Results:





Attempt 4:

a) Preprocessing:

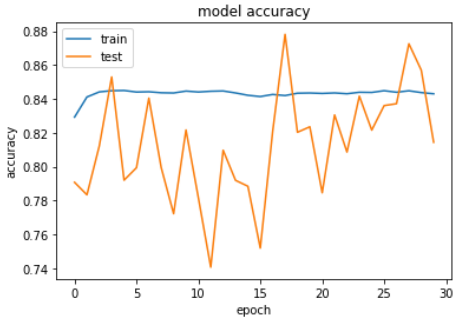
a. Under-Sampling the majority class to only 0.1 of the its size
b. Feature Engineering applied
c. Class weights adjusted to favor Class 1 3 times Class 0

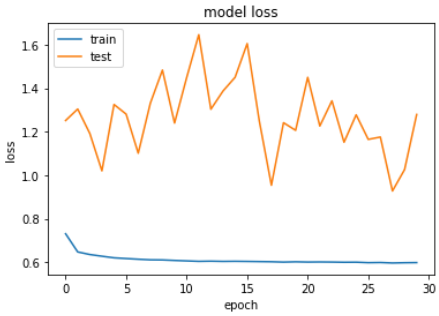
b) Model Implementation

- Number of Epochs: 30

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, 161, 100)          4416000
_________________________________________________________________
dense_60 (Dense)             (None, 161, 16)           1616
_________________________________________________________________
max_pooling1d_12 (MaxPooling (None, 80, 16)            0
_________________________________________________________________
flatten_22 (Flatten)         (None, 1280)              0
_________________________________________________________________
dense_61 (Dense)             (None, 2)                 2562
=================================================================
Total params: 4,420,178
Trainable params: 4,178
Non-trainable params: 4,416,000
_________________________________________________________________
None
```

c) Results:





Attempt 5:

a) Preprocessing:

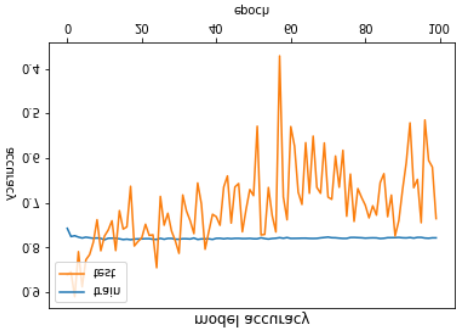
a. Under-Sampling the majority class to only 0.1 of the its size
b. Feature Engineering applied
c. Class weights adjusted to favor Class 1 3 times Class 0

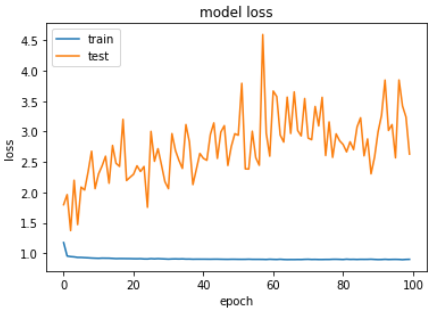
b) Model Implementation

- Number of Epochs: 100

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_5 (Embedding)      (None, 161, 100)          4416000
_________________________________________________________________
dense_65 (Dense)             (None, 161, 64)           6464
_________________________________________________________________
dropout_43 (Dropout)         (None, 161, 64)           0
_________________________________________________________________
dense_66 (Dense)             (None, 161, 128)          8320
_________________________________________________________________
max_pooling1d_14 (MaxPooling (None, 40, 128)           0
_________________________________________________________________
flatten_24 (Flatten)         (None, 5120)              0
_________________________________________________________________
dense_67 (Dense)             (None, 2)                 10242
=================================================================
Total params: 4,441,026
Trainable params: 25,026
Non-trainable params: 4,416,000
_________________________________________________________________
None
```

c) Results





Attempt 6:

a) Preprocessing:

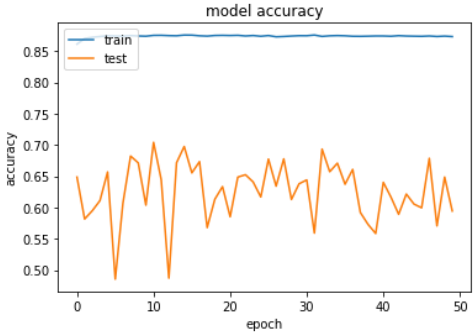
a. Under-Sampling the majority class to only 0.1 of the its size
b. Feature Engineering applied

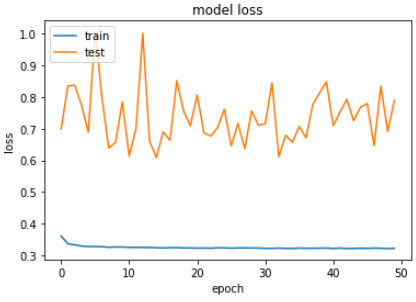
b) Model Implementation

- Number of Epochs 50

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_5 (Embedding)      (None, 161, 100)          4416000
_________________________________________________________________
dense_85 (Dense)             (None, 161, 256)          25856
_________________________________________________________________
dropout_60 (Dropout)         (None, 161, 256)          0
_________________________________________________________________
flatten_34 (Flatten)         (None, 41216)             0
_________________________________________________________________
dense_86 (Dense)             (None, 1)                 41217
=================================================================
Total params: 4,483,073
Trainable params: 67,073
Non-trainable params: 4,416,000
_________________________________________________________________
None
```

c) Results





Justification

Attempt No.	Naïve Bayes F1-Score	CNN F1-Score
1	0.54	0.49
2	0.42	0.16
3	0.54	0.5
4	0.54	0.51
5	0.54	0.52
6	0.54	0.44

The CNN model performance was not as expected. It did not converge to an optimal solution, and has not yielded in a better solution than Naïve Bayes. Most of the accuracies are close enough however it is not at all reliable. Perhaps, fine-tuning the parameters more could possible enhance the results, however, the problem lies in the the dataset, the model might have been over-fitting because it is not fed enough data, also because of the class imbalance problem we have discussed earlier.

V. Conclusion

Free-Form Visualization

The main bottleneck faced in this project in my opinion was the imbalanced data obtained by the dataset which was 15:1, however, it was later mitigated by under-sampling the data and achieving 2 binary classes close to each other, forming a somewhat balanced training set, with lower class to class ratio.

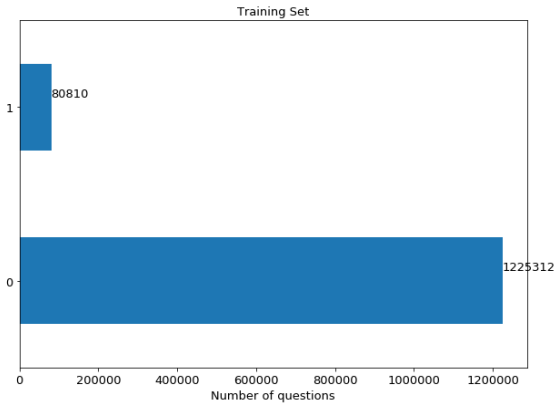


Figure 1 Imbalanced original training set

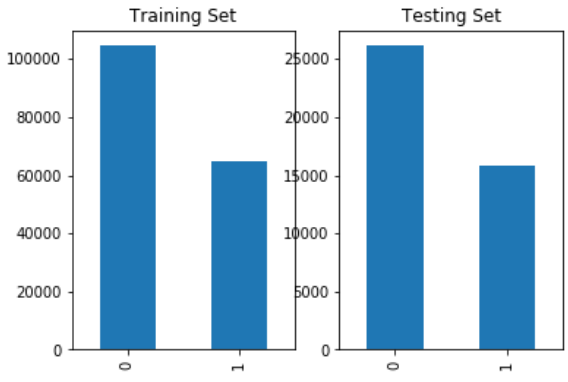


Figure 2 Under-sampled training set and test set

Reflection

In summary, the objective of this project was to draw a comparison between regular supervised learning text classifiers and a CNN approach, because a CNN approach can be considered a model that can draw relation in textual data through proper vector space. Also, the objective was to explore other vectorization techniques than Bag of Words or TFIDF to feed our classifiers.

What I found interesting was how much the training set used and the classes ratio can contribute to the overall classifier performance, and it has a larger effect than tuning the hyper-parameters, like in attempt 2, when we adjusted with the training set and increased the gap between the classes, it yielded the worst performance, even though we used nearly the same network architecture. Also, methods like SMOTE can help with imbalanced data, however, it greatly influences the performance of the model in terms of speed and time, and it's better to use this oversampling approach with small imbalanced data.

Even though the final model did not generalize as expected, these findings can be seen as an active approach to find a model that can generalize better.

Improvement

The project can use a lot of improvements to generalize to a better model, in the future I would like to explore the following:

- Find a better technique to balance between the classes of data than under-sampling and oversampling
- Explore the possibility of an RNN approach and test how the model compares to a standard supervised learning model and a CNN model
- Improve the feature engineering step by formatting the data more and extracting sentiment relationships between sentences in class 0 and others in class 1.

---

[i] Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, (pp. 1746–1751.).